View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by KITopen

Unified Domains and Abstract Computational Structures

J. CALMET, K. HOMANN and I.A. TJANDRA

Universität Karlsruhe
Institut für Algorithmen und Kognitive Systeme
Am Fasanengarten 5; Postfach 6980; D-76128 Karlsruhe; Germany

Abstract. This paper introduces a formalism to specify abstract computational structures (ACS) of mathematical domains of computation. This is a basic step of a project aiming at designing an environment for symbolic computing based upon knowledge representation and relying, when needed, on AI methods.
We present a method for the specification of these ACS's which is embedded in the framework of algebraic specifications and of unified domains. The first part of this paper deals with the theoretical solution of this specification problem. The second part reports on the implementation in the hybrid knowledge representation system MANTRA.

Keywords: Abstract Computational Structures, Unified Domains, Hybrid Knowledge Representation System.

1 Introduction

Mathematical domains of computations, such as finite groups, polynomial rings or finite fields for instance, are inherently modular. Furthermore, there exist inter-relationships among these domains. It is thus meaningful to design an environment for symbolic computing within the framework of the theory of algebraic specification [Gu 75], [Gu], [Go,Bu]. An algebraic specification basically introduces constants and operators together with properties that have to be imposed on their intended interpretation, or class of interpretations. This has several pragmatic benefits, such as the re-use of subspecifications within a specification taking into account the dependency relationship between particular specification modules.

Because we are mainly concerned with executable specifications, it turns out that the framework of *unified domains*[1] [Mo86] developed from the framework of order-sorted domains, cf. section 2, is appropriate to handle mathematical domains of computation. Furthermore, when defining the semantics of vertical or *inclusion polymorphism* in a Computer Algebra System one routinely deals with operations that map sorts to sorts, for instance mapping two sorts to their union or to their subset. This is well supported by unified domains since there is

[1] originally called unified algebras

a unified treatment of the *elements* of an abstract computational structure and their classifications into *sorts*. Thus the operations of a unified domain may take sorts and/or elements as arguments, and give sorts or elements as results [Mo86]. This framework improves much our first approach to this problem [Ca-Tj91a].

This paper outlines how to specify abstract computational structures (ACS's) in the framework of unified domains. The specifications are represented internally using the language that is embedded in the knowledge representation system MANTRA. We have specifically designed MANTRA [Ca et al. 91b] as a suitable environment for symbolic mathematical computation. Many aspects must be considered when planning such an environment. Among them is the capability to handle inclusion polymorphism, which is fundamental for symbolic mathematical computations. Also, it is known that no universal algorithm does exist for problems such as type inference for mathematical domains when properties are considered or canonical forms for mathematical expressions. Possible solutions consist in relying on methods from AI. Such problems were among the many motivations to design our environment.

Basically, the procedure of performing a specification can be depicted as follows: Having recognized the syntax correctness of an input (a specification) according to the underlying specification language, a parse tree is given to a program transformer that generates code, possessing the intended semantics, for the language of MANTRA. The code will be executed by MANTRA by invoking its inference engine.

The paper is organized as follows. As the framework of unified domains is developed from the framework of *order-sorted domains* we introduce in section 2 some basic notions and notations of order-sorted domain [Ba-Wo][Hu,Op]. Section 3 introduces the notion and notations of unified domains which is a specialization of order-sorted domains. Section 4 deals with an introduction to MANTRA. We outline only those features of MANTRA that we need in the sequel. The internal representation of the specification of ACS's is described in section 5. Finally, some concluding remarks are given in the last section.

2 Basic Notions and Notations

$\Sigma = \langle \mathcal{S}, \Omega \rangle$ is an order-sorted signature consisting of $\mathcal{S}$, a finite set of order-sorted sorts, and Ω, the set of operator symbols. Ω is the union of pairwise disjoint subsets of $\Omega_{w,s}$, the sets of operator symbols with argument sorts $w \in \mathcal{S}^*$, i.e. $w = s_1 \times \cdots \times s_n$ for $n \geq 0$ and $s_i \in \mathcal{S}$, and range sort $s \in \mathcal{S}$.

A domain $\mathcal{D}$ with the signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$, also called Σ-domain, is a pair $\mathcal{D} = \langle \mathcal{U}, \mathcal{F} \rangle$ such that $\mathcal{U}$ is an $\mathcal{S}$-indexed family of sets and $\mathcal{F}$ is a Ω-indexed family of functions with: (i) $\mathcal{U}_s$ are sets for all $s \in \mathcal{S}$, called the universe of sorts s in the domain. (ii) $\mathcal{F}_f \in \mathcal{U}_s$ if $f \in \Omega_{\lambda,s}$ and $s \in \mathcal{S}$, called constants of $\mathcal{D}$ of sort s. (iii) $\mathcal{F}_f \in \mathcal{U}_{s_1} \times \cdots \times \mathcal{U}_{s_n} \rightarrow \mathcal{U}_s$, functions for all operator symbols f, if $f \in \Omega_{s_1 \times \cdots \times s_n, s}$ with $n \geq 1$, $s, s_i \in \mathcal{S}$ and $i = 1, \cdots, n$.

Let $\Sigma = \langle \mathcal{S}, \Omega \rangle$ be a signature and X_s for each $s \in \mathcal{S}$ a set of variables of sort s. We assume that the variables of distinct sorts are distinct and that no

variable is a member of Ω. The union $X = \cup_{s \in \mathcal{S}} X_s$ is called the set of variables with respect to Ω.

The sets $T_{s,\Omega}(X)$ of terms of sort s are inductively defined as follows: (i) $X_s \cup \Omega_{\lambda,s} \subseteq T_{s,\Omega}(X)$ (*basic terms*), (ii) $f(t_1, \cdots, t_n) \in T_{s,\Omega}(X)$ (*composite terms*) for all operator symbols $f \in \Omega_{s_1 \times \cdots \times s_n, s}$ and all terms $t_i \in T_{s_i,\Omega}(X)$ $i = 1, \cdots, n$, (iii) there are no further terms of sort s in $T_{s,\Omega}(X)$. The set $T_{s,\Omega}$ of terms without variables of sort s, also called *ground terms* of sort s, is defined for the empty set $X = \emptyset$ of variables as follows: (iv) $T_{s,\Omega} = T_{s,\Omega}(\emptyset)$. The set of terms $T_\Sigma(X)$ and the set of terms without variables T_Σ are defined as follows: (v) $T_\Sigma(X) = \bigcup_{s \in \mathcal{S}} T_{s,\Omega}(X)$ with $s_1, s_2 \in \mathcal{S},\ s_1 \sqsubseteq s_2,\ t \in T_{s_1,\Omega}(X) \Rightarrow t \in T_{s_2,\Omega}(X)$ and $T_\Sigma = \bigcup_{s \in \mathcal{S}} T_{s,\Omega}$ with $s_1, s_2 \in \mathcal{S},\ s_1 \sqsubseteq s_2,\ t \in T_{s_1,\Omega} \Rightarrow t \in T_{s_2,\Omega}$. This also implies that each sort (type) inherits all terms possessed by its subsorts (types).

Let T_Σ be the set of terms of signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and $\mathcal{D}$ a Σ-domain. The *interpretation* $\varphi : T_\Sigma \to \mathcal{D}$ is recursively defined as follows: (i) $\varphi(f) = \mathcal{F}_f$ for all $f \in \Omega_{\lambda,s}$, $s \in \mathcal{S}$, (ii) $\varphi(f(t_1, \cdots, t_n)) = \mathcal{F}_f(\varphi(t_1), \cdots, \varphi(t_n))$ for all $f(t_1, \cdots, t_n) \in T_\Sigma$.

Given a set of variables X for $\Sigma = \langle \mathcal{S}, \Omega \rangle$, a Σ-domain $\mathcal{D}$ and an assignment $\psi : X \to \mathcal{D}$ with $\psi(x) \in \mathcal{U}_s$ for $x \in X_s$ and $s \in \mathcal{S}$, the *extension* $\xi : T_\Sigma(X) \to \mathcal{D}$ of the assignment $\psi : X \to \mathcal{D}$ is recursively defined as follows:

(i) $\xi(x) = \psi(x)$ for all variables $x \in X$.
$\xi(f) = \mathcal{F}_f$ for all $f \in \Omega_{\lambda,s}$, $s \in \mathcal{S}$.
(ii) $\xi(f(t_1, \cdots, t_n)) = \mathcal{F}_f(\xi(t_1), \cdots, \xi(t_n))$ for all $f(t_1, \cdots, t_n) \in T_\Sigma(X)$

Given a signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and variables X with respect to Σ. A triple $p = \langle X, L, R \rangle$ with $L, R \in T_{s,\Omega}(X)$ for some $s \in \mathcal{S}$ is called a property of sort s with respect to Σ. The property $p = \langle X, L, R \rangle$ is said to be *valid* in a Σ-domain $\mathcal{D}$ if for all assignment $\psi : X \to \mathcal{D}$ we have $\xi(L) = \xi(R)$. *Ground properties* are properties $p = \langle X, L, R \rangle$ with $X = \emptyset$. In this case L and R are ground terms.

An *abstract computational structure* $ACS = \langle \Sigma, \mathcal{P} \rangle$ consists of a signature $\Sigma = \langle \mathcal{S}, \Omega \rangle$ and $\mathcal{P}$, a set of properties with respect to Σ. Based on other known ACS's we can construct a new ACS by integrating all operators and properties possessed by the known ACS's and by adding additional operators and properties into the new one. This implies that each ACS inherits all operators and properties possessed by the ACS on which it is based.

3 Unified Domains

Initially, we define signatures and properties for unified domains by specializing the notation for abstract computational structures and domains as described in section 2.

A unified signature Σ^u is a *homogeneous first-order signature*. In contrast to order-sorted signatures there is only one sort, $\mathcal{S}^u$, instead of a set of order-sorted sorts[2]. Let Σ^u be a pair $\langle \mathcal{S}^u, \Omega^u \rangle$, where $\Omega^u \supseteq \Omega^{u0} = \{\bot, |, \&\}$. $\bot$, $|$ and

[2] As we shall see below, we still can simulate an order-sorted signature using a unified domain, since the carrier of a unified domain is a distributive lattice.

& represent the bottom of a lattice, the joint and meet operations on lattices, respectively. As we deal with homogeneous first-order signatures we can write $\Omega^u = \{\Omega^u_n \mid n \geq 0\}$, where Ω^u_n is the set of operator symbols of arity n.

Let X be a set of variables, disjoint from Ω^u. A Σ^u-unified property is a universal Horn Clause involving equalities with variables from X, operator symbols form Ω^u and the binary predicate symbols **Identical-to**, **Subsumes** and **Element-of**.

A Σ^u-unified domain $\mathcal{D}$ is a homogeneous Σ^u-domain –cf section 2– such that:

(i) $|\mathcal{D}|$ is a distributive lattice with $\_|\__{\mathcal{D}}$ as join, $\_\&\__{\mathcal{D}}$ as meet and $\bot_{\mathcal{D}}$ as bottom. Let $\mathbf{Subsumes}_{\mathcal{D}}$ be the partial order of the lattice.
(ii) There is a distinguished subset of incomparable values, $\mathcal{E}_{\mathcal{D}} \subseteq| \mathcal{D} |$.
(iii) For each $f \in \Omega^u$, the function $f_{\mathcal{D}}$ is monotone with respect to $\mathbf{Subsumes}_{\mathcal{D}}$.

The intended interpretation of the binary predicate symbols, in a unified domain $\mathcal{D}$ is as follows:

- x **Identical-to** y :⇔ x is identical to y
- x **Subsumes** y :⇔ x $\mathbf{Subsumes}_{\mathcal{D}}$ y
- x **Element-of** y :⇔ x $\in \mathcal{E}_{\mathcal{D}}$ and x $\mathbf{Subsumes}_{\mathcal{D}}$ y

4 MANTRA

The *MANTRA* system [Ca et al. 91b] is a hybrid system with the following characteristics: All features are semantically motivated and all the inference algorithms involved are decidable. The decidability requirement has been met with the adoption of a four-valued semantics based on the works of Patel-Schneider [PS85], Frisch [Fr] and Thomason et al. [Th et al.]. This semantics is used throughout the system and ensures that it is semantically consistent.

The language of MANTRA can be thought of as an abstract data type allowing the creation and manipulation of knowledge bases. The *knowledge bases* consist of a set of knowledge base partitions, each associated to an independent formalism. The division of the language into several formalisms has two advantages: The computability problems associated to each formalism can be solved independently and the integration of new formalisms to the system is facilitated.

Each formalism is characterized by a set of definitions and a set of questions. The *definitions* allow the storage of knowledge into the knowledge bases and the *questions* allow the interrogation of these knowledge bases. Definitions are used to store knowledge only into the partition associated to the formalism, but questions can be directed to this partition or to a combination of two or more partitions of the knowledge base. The language is based on a new architecture, figure 1, consisting of three levels: The epistemological level, the logical level and the heuristic level.

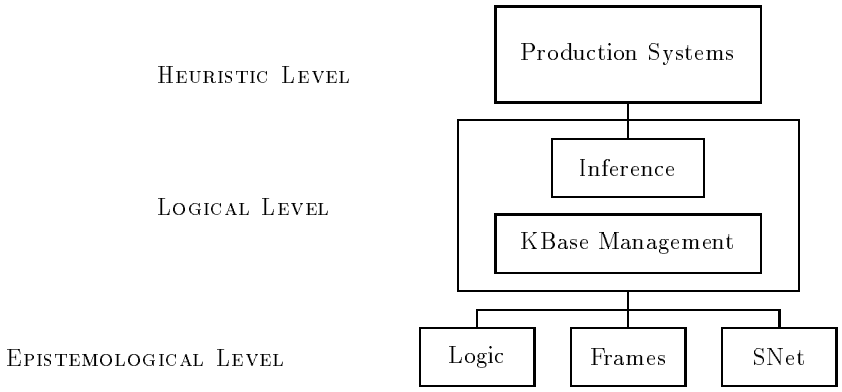

Fig. 1. The architecture of *MANTRA*

The first level consists of three modules: an assertional module, based on a decidable first-order logic language [PS85], a frame module, based on the terminological box of Krypton [Br et al.], and a semantic network module, providing inheritance with exceptions [Et]. These modules offer original features with respect to previously existing systems. The primitives of the modules of this first level define the epistemological primitives of the language. These primitives are not complete expressions of the language but are used as parameters for the Ask and Tell primitives of the logical level.

The primitives of the *assertional module* correspond to the usual operators of the first-order logic languages, but the meaning of these operators is based on a four-valued semantics.

This module is intended to be used to represent a terminology by means of concepts, the categories of objects, and relations, the properties of objects. These categories are described by restricting the values of the properties of the objects forming them. The notion of relations is an extension of the notion of roles, usually used in terminological languages. Roles are binary relations and relations are arbitrary n-place relations.

The terminological language embedded into the system has some additional characteristics usually not possessed by other terminological languages or hybrid systems: (i) It possesses a rich set of primitives, including disjunction and negation of both concepts and relations, (ii) It provides special symbols for the universal concept and for the bottom concept as well as for the universal relation and for the bottom relation and (iii) It includes tests for subsumption and for equality between concepts and between relations.

This module manipulates the notions of classes and hierarchies. The hierarchies can be explicitly created by defining links among classes. Two types of links are provided: *Default* links and *Exception* links. The hierarchies are used as

inheritance paths between classes. The main inference procedure of this module calculates the *Subclasses* relation taking into account the explicit exception.

The second level introduces the notion of knowledge base. The language can be thought of as an abstract data type whose access functions are the primitives of this level. These primitives use the primitives of the first level as parameters. Two types of primitives are provided, **Tell** and **Ask**. These primitives are used, respectively, to store facts and to interrogate knowledge bases. The Ask primitives are defined in such a way that new facts can be inferred from evidence provided by the knowledge acquired only by one or by a combination of several of the first level modules.

Finally, the third level consists of primitives allowing the definition of production systems according to several configurations [Le-De]. These production systems are used to automatically manipulate knowledge bases according to heuristic rules. The rules used in these production systems are formed by queries to the knowledge bases defined in the logical level. Conflict resolution strategies and control strategies can be explicitly chosen by special primitives. The idea underlying the heuristic level is to allow the introduction of ad hoc rules in the inference process. These rules can directly introduce domain knowledge in the knowledge bases or they can specify strategies for the utilization of the logical level Ask and Tell primitives. This level enables thus to design expert systems.

This brief overview of *MANTRA* covers only those features which are used in the sequel.

5 Representation of ACS's using MANTRA

As properties are embodied by Horn Clauses, the class of Ω^u-unified domains that satisfy $\mathcal{P}$ has an initial domain [Go et. al.]. The proof of this theorem can be found in [Mo89].

If we impose initial constraints on Σ^u-unified domains then we are capable of specifying ACS's using an Σ^u-unified domain.

We describe the syntax of the language for algebraic specifications of abstract computational structures and its intended interpretation by means of some examples, cf. figure 2.

The specification of an ACS has two parts:

(a) A header, containing its name and
(b) A body, containing its signature that is divided into the following parts:
   (i) **Based-on**: This declaration is to indicate those abstract computational structures that are inherited by the ACS being specified.
   (ii) **Parameter**: This declaration is used to specify an ACS possessing particular ACS's as parameters, e.g. ACS Module has parameter ACS Rg and based on additive abelian group.
   (iii) **Sort**: To declare its new sorts. Subsort relationships are declared using the notations as introduced above, e.g. using | and &.

```
ACS Semi-Group {
        Based-on set;
        Sort SG;
        Operators _ f _ :: Elt - > Elt;
        Initial-Properties
        !V x,y,z : x Element-of Elt, y Element-of Elt, z Element-of Elt
                => (x f y) f z = x f (y f z)}
ACS Monoid {
        Based-on Semi-Group;
        Sort Mo, ne Element-of Elt;
        Initial-Properties
        !V x: x Element-of Elt
                => ne f x = x}
ACS Group {
        Based-on Monoid;
        Sort Gr;
        Operators _ inv _ :: Elt - > Elt;
        Initial-Properties
        !V x: x Element-of Elt
                => inv(x) f x = ne}
ACS Abelian-Group {
        Based-on Group;
        Sort AG;
        Initial-Properties
        !V x, y: x Element-of Elt, y Element-of Elt
                => x f y = y f x}
ACS Rg {
        Based-on Semi-Group(rename: (f,×), (ne,1)),
                Abelian-Group(rename: (f,+), (ne,0), (inv, −));
        Sort R;
        Initial-Properties
        !V x, y, z: x Element-of Elt, y Element-of Elt, z Element-of Elt
        => x×(y+z) = (x × y) + (x × z)
        !V x, y, z: x Element-of Elt, y Element-of Elt, z Element-of Elt
                => (y+z)×x = (y × x) + (z × x)}
```

Fig. 2. Examples of some basic ACS

(iv) **Operator-Symbols**: In this part all new operator symbols are declared. :: and − > are special symbols used to define the functionality of an operator.

(v) **Initial-Properties**: To declare properties of the operators. ***!V*** represents the universal quantifier.

The use of **Based-on** declaration permits the re-use of parts of certain specifications in another specification, which is very convenient to design abstract computational structures for mathematical domains according to their inher-

ently modular structure. The intended interpretation of this declaration is the inheritance of sorts – e.g. implicitly, in ACS Monoid the sort becomes Mo | SG where Mo and SG denote the sorts of Monoid and Semi Group, respectively, and so on– , operator symbols and initial properties possessed by the ACS embedded in this specification. Inherited symbols are not changed unless we rename explicitly old symbols using **rename**, e.g. in the ACS **Rg** below.

We use the mixfix notation for operator symbols according to the positions of place holders.

Recalling that classifications of elements into sorts are represented directly as values in the corresponding carriers, we define constants or nullary functions using **Element-of**.

We represent initial properties as rules, rather than merely writing them as comments, since we use them to simplify terms. Generally, a set of properties is not complete in the sense that terms can not canonically be simplified using these properties. Therefore we classify properties into two classes: Initial properties and learned properties. The method to complete a set of properties [Ca-Tj90] is beyond the scope of this paper. In this paper, we merely take into account initial properties.

The main principle of representing an ACS can be described as follows:

(i) The signature of an ACS, as shown in figure 2, is represented as frames. The relationships among ACS's are represented as semantic networks wherein each node corresponds to a particular frame, e.g. Semi-Group-Operators and Monoid-Operators. The link specifies which objects are inherited by which ones within a particular hierarchy, e.g. in the hierarchy **Based-on** there is a link from Semi-Group-Operators to Monoid-Operators meaning that, according to the examples, Semi-Group-Operators is inherited by Monoid-Operators.

(ii) The initial properties of an ACS are represented as rules and they are stored in a rule base. To identify the rule base we introduce a relation **rule-base**, possessed by an ACS frame, whose value is the identifier of the corresponding rule base.

Thus, an ACS is represented by a frame having a unique identifier, **id**. This frame possesses a relation **rule-base** and subsumes the frame *sort*, i.e. the frame identifier coincides with the sort identifier, and the frame **id**-Operators.

The frame *sort* subsumes another frames if the relation **Element-of** occurs in the sort declaration, e.g. in ACS Monoid: ne **Element-of** Mo. The frame **id**-Operators subsumes frames, representing each operator symbol, possessing the relation **domain** and the relation **codomain** specifying the domains of an operator and its codomain, respectively, cf. figure 3.

Having recognized the declaration **Based-on** within an ACS the program transformer build or modify the hierarchy (semantic network) *based-on* according to the specification. This is illustrated by figure 4.

The Interrogation of a knowledge base need a hybrid inference taking into account frames and semantic network simultaneously, e.g. using the primitive

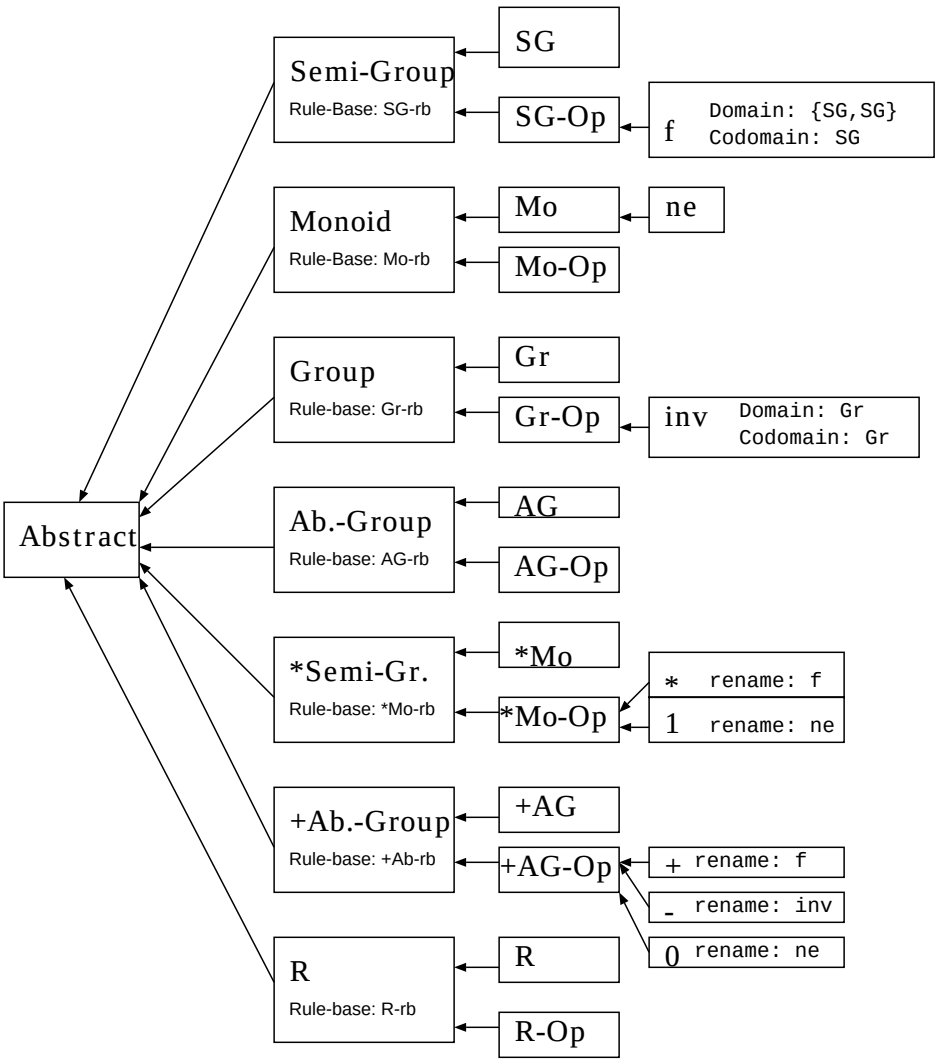

Fig. 3. Frame Structure

```
(ASK 'KB '(FROM-SNET-FRAME ........))
```

Basically, mathematical domains of computations are models of ACS's. According to our approach they can be regarded as unified domains, since initial constraints are imposed on unified domains. We shall outline this part briefly.

The specification of a domain is quite similar to that of an ACS. The semantic specification of operations plays a crucial role and we adopt action semantics to specify the semantic functions.

Generally, the specification of a unified domain has two parts: (a) a header,

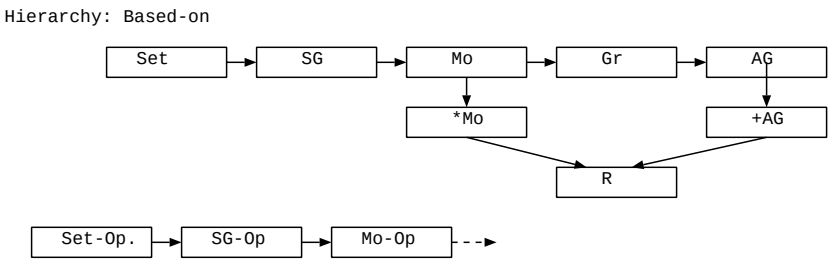

Fig. 4. The hierarchy *based-on*

containing its name and (b) a body as defined for ACS's but without initial properties and with two additional declaration: (i) **Model**, identifying the corresponding ACS and (ii) **Semantic-Equations**. The given specification will be transformed into the language of MANTRA where semantic equations will be represented accordingly by rules in such a way that they can be performed by the inference engine.

6 Conclusion

We have outlined the framework of Unified Domains and its application to specify abstract computational structures of mathematical domains of computation.

Unified domains make intensive use of nondeterministic operations, which correspond to union of classifications. We can use unified domains not solely to specify but, furthermore, it underlies the inclusion partial ordering which is preserved by all the other operations. This is the major advantage of using unified domains since it amounts to an easy implementation of vertical or inclusion polymorphism. We deal with this kind of polymorphism since we design an environment for symbolic computing.

In order to represent such domains we make use of the hybrid knowledge representation system MANTRA. The characteristic features of MANTRA consist in its decidable algorithms and its unified semantics. This ensures the security of its performance in the sense that each algorithm terminates. The hybrid inference mechanisms of MANTRA are used to process knowledge bases.

To design an executable or operational semantic function for mathematical domains of computation we make use of Action Semantics. In Action Semantics the semantic functions are defined inductively by *semantic equations*, called actions. These actions may be regarded as algebraic equations, but their *well-foundedness* is essential and they have a more operational nature than, for instance, the higher-order functions used as denotations in denotational semantics [St]. The details of this approach is out of the scope of this paper. We have merely depicted the major principle briefly.

Finally, it can be summarized that Unified Domains have made a significant contribution to specifying ACS's and a hybrid knowledge representation can be used to represent such domains.

References

[Ba-Wo] Bauer, F.L. and Woessner, H., *Algorithmische Sprache und Programmentwicklung(2nd Edition)*, Springer-Verlag Berlin Heidelberg New York Tokyo, 1984.

[Br et al.] Brachman, R.J., Gilbert, V.P., Levesque,H.J., *An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON*, Proceedings of IJCAI 9, pp. 532–539, 1985.

[Ca-Tj90] Calmet,J., Tjandra,I.A., *Learning Complete Computational Structures*, in Emrich et al (Eds), 5th International Symposium on Methodologies for Intelligent Systems (Selected Papers), Knoxville – USA, October 24 – 27, 1990, pp. 63 – 71, ICAIT.

[Ca-Tj91a] Calmet,J., Tjandra,I.A, *Representation of Mathematical Knowledge*, Z. W. Ras et al.. (Eds.) Proceedings of the 6th International Symposium on Methodologies for Intelligent Systems, Charlotte USA, October 16 – 19, 1991, Springer-Verlag.

[Ca et al. 91b] Calmet,J., Bittencourt,G., Tjandra, I.A., *MANTRA: A Shell for Hybrid Knowledge Representation*, Proceedings of the third International conference on Tools for Artificial Intelligence, San Jose USA, November 5 – 8, 1991, IEEE Computer Society Press.

[Et] Etherington,D.W., *On Inheritance Hierarchies with Exceptions*, Proceedings of AAAI-83, pp. 104–108, 1986.

[Fr] Frisch,A.M., *Knowledge Retrieval as Specialized Inference*, Report No.214, Department of Computer Science, University of Rochester, May 1987.

[Go,Bu] Goguen, J.A. and Burstall, R.M., *Introducing Institutions*, in Clarke, E. and Kozen, D. (Eds.), Proceedings Logics of Programming Workshop, Springer-Verlag, 1984.

[Go et. al.] Goguen, J.A., Thatcher, J.W. and Wagner, E., *An Initial Algebra Approach to the Specification Correctness and Implementation of Abstract Data Types*, in Yeh, R.T. (Ed.), Current Trends in Programming Methodology IV, Prentice Hall, 1978.

[Gu 75] Guttag, J. : *The Specification and Application to Programming of Abstract Data Types*, University of Toronto, Department of Computer Science, Ph. D. Thesis, Report CSRG-59 ; 1975.

[Gu] Guttag,J.V., *The Algebraic Specification of Abstract Data Types*, Acta Informatica 10, 27–52, Springer-Verlag, 1978.

[Hu,Op] Huet, G., Oppen D.: *Equations and Rewrite Rules: a survey*; in Book R., editor, Formal Language Theory: Perspective and Open Problems, Academic Press, 1980.

[Le-De] Lenat, D.B., McDermott,J., *Less Than General Production Systems Architectures*, Proceedings of IJCAI 5, pp. 928–932, 1977.

[Mo86] Mosses, P.D., *Unified Algebras and Action Semantics*, in Monien, B. and Cori, R. (Eds.), Proceedings of 6th Annual Symposium on Theoretical Aspects of Computet Science, Springer-Verlag, 1986.

[Mo89] Mosses, P.D., *Unified Algebras and Institutions*, in Proceedings IEEE-Logics in Computer Science, IEEE-Press, 1989.

[PS85] Patel-Schneider, P.F., *A Decidable First-Order Logic for Knowledge Representation*, Proceedings of IJCAI 9, pp. 455–458, 1985.

[St] Stoy, J.E., *Denotational Semantics: The Scott-Strachey Approach toProgramming Language Theory*, the MIT Press, 1977.

[Th et al.] Thomason, R.H, Horty, J.F. and Touretzky, D.S., *A Calculus for Inheritance in Monotonic Semantic Nets*, Technical Report CMU-CS-86-138, Computer Science Department, carnegie mellon University, Pittsburgh, PA, 1986.

This article was processed using the LaTeX macro package with LLNCS style